

AD-A039 742

FEDERAL COBOL COMPILER TESTING SERVICE WASHINGTON D C
HYPO-COBOL -- A COBOL SUBSET FOR MINICOMPUTERS. (U)
MAY 77 M M COOK, G N BAIRD

F/G 9/2

UNCLASSIFIED

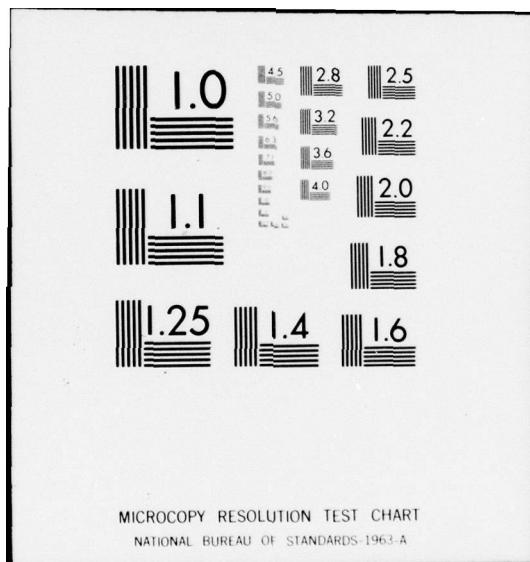
FCCTS/TR-77/16

NL

1 OF 1
ADA039 742

END

DATE
FILMED
6-77



ADA 039742

FILE COPY
DDC FILE COPY
FILE NO.

See 113
113
b5
7

HYPO-COBOL -- A COBOL SUBSET FOR MINICOMPUTERS

Margaret M. Cook
George N. Baird

ADPE Selection Office
Department of the Navy
Washington, D. C. 20376

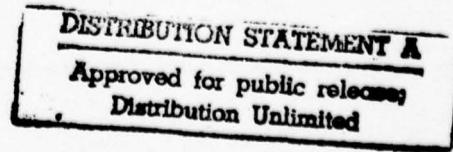


ABSTRACT

The HYPO-COBOL language is a COBOL subset designed for implementation in minicomputer and time-sharing environments which impose resource limitations. The HYPO-COBOL language definition was prompted by the need for a COBOL subset in a small scale package which could provide basic capabilities, yet still not make extreme demands on system resources during compilation. It is intended as an alternative to full COBOL for those users who do not require the complete COBOL language.

The base document from which the specifications for the HYPO-COBOL language were developed was American National Standard Programming Language COBOL, X3.23-1974. The HYPO-COBOL language consists of subsets of the Nucleus and five functional processing modules: Table Handling, Sequential I-O, Relative I-O, Debug, and Inter-Program Communication.

This paper presents the background and reasons for the definition of the HYPO-COBOL language, and the design features and rationales of the language. A HYPO-COBOL Compiler Validation System for determining compliance to the specifications has been developed and is also described.



INTRODUCTION

Background

In July 1973, the Department of the Navy signed a contract for an indefinite quantity of small scale minicomputer systems. The purpose of the procurement was to allow a user with a small application requirement to order a system from the contract for ninety-day delivery, and thereby avoid the lengthy delay in the normal procurement process. Some of the minicomputer acquisitions under this contract have gone to activities involved in business data processing, but the software available with this system was FORTRAN, BASIC, an assembly language and other generalized utility software. The business applications had to be adapted so that FORTRAN or BASIC could handle the problem.

The need for a business type language for minicomputers was recognized by Captain Grace Hopper (USNR) as early as 1971 when she directed an effort by members of the Navy Programming Languages Section to produce a scaled down version of the COBOL language and a prototype compiler for implementation on minicomputer systems. The purpose in developing the prototype compiler was to determine the implementability of the specifications on a minicomputer.

The GML Minicomputer Review 1975¹ indicates that there are presently 28 minicomputers available from mainframe manufacturers which have a COBOL compiler. There are also a number of COBOL compilers which have been developed by software houses for some of the currently available minicomputers which are not included in the GML Review totals.

White Section
Black Section

BY.....
DISTRIBUTION/AVAILABILITY CODES

Dist. AVAIL. and/or SPECIAL

A			
---	--	--	--

The majority of these COBOL compilers do not meet the requirements of a minimum COBOL implementation as defined in American National Standard Programming Language COBOL, X3.23-1974². The implementors of compilers have in effect defined their own unique subsets of COBOL. Because of this, COBOL programs written for one minicomputer would probably not be usable on another minicomputer system without an unreasonable amount of modification.

A second Navy procurement for an indefinite quantity of mini-computer systems was approved in October 1975 and contained a requirement for a business type programming language. COBOL was chosen as the language but there was a question as to what level of COBOL should be required. The COBOL requirement was stated as the Low Intermediate Level of Federal Standard COBOL⁴. In order to encourage competition and at the same time not place an unreasonable requirement on vendors, an alternative to the Low Intermediate level of COBOL was offered. The COBOL requirement then became a choice of providing Low Intermediate COBOL or a proper subset of Low Intermediate COBOL defined by the Navy. The alternative offered was the HYPO-COBOL language³ which is the subject of this paper.

A New COBOL Subset

When the requirement for a small scale COBOL subset was identified, the existing alternatives for COBOL subsets were examined. X3.23-1974 defines the minimum American National Standard COBOL to be level 1 of the Nucleus, Table Handling, and Sequential I-O modules. Already this

imposed a difficulty for a minimum subset since there are features in these three modules which would be a burden for a system with limited resources. Thus, a subset formed from any combination of the complete modules as defined in X3.23-1974 would not be satisfactory.

Federal Information Processing Publication 21-1, Federal Standard COBOL⁴ adopted American National Standard Programming Language COBOL, X3.23-1974 as a Federal Standard. FIPS PUB 21-1 defines four subsets from the X3.23-1974 specifications. The Low Level Federal Standard COBOL contains level 1 of Nucleus, Table Handling, and Sequential I-O, the same as the minimum implementation defined in X3.23-1974. The Low Intermediate level of Federal Standard COBOL consists of the Low Level Federal COBOL and level 1 of the Relative I-O, Library, Segmentation, Debug, and Inter-Program Communication modules. Both of these subsets are too demanding and tend to ignore the functional capability of most computing systems in the "mini" range.

The new COBOL subset produced by the Department of the Navy defined subsets for Nucleus and the Table Handling and Sequential I-O modules. Minicomputer and time-sharing systems support additional features besides those included in the minimum level. For this reason, the other functional processing modules in X3.23-1974 were examined, and subsets of the Relative I-O, Debug, and Inter-Program Communication modules were included in the HYPO-COBOL language.

Why "HYPO"?

Many people have inquired why the subset is called HYPO-COBOL rather than MINI-COBOL. The prefix "mini" was rejected because it ties

the new COBOL subset exclusively to minicomputer applications. The subset is intended for any limited resources environment, not just minicomputers. The search for a different prefix produced "hypo".

The American Heritage Dictionary⁵ defines the prefix "hypo" to mean "below or beneath", or "at a lower point". The aim in defining the HYPO-COBOL language was to obtain a COBOL subset which is "at a lower point" than the full COBOL language, but still retains the minimum useful features of COBOL.

Design Criteria

The main purpose of the HYPO-COBOL language definition was to reduce the size of the COBOL compiler and its related work areas. The language was to maintain a low level of sophistication, but still provide all the basic functions necessary to support general minicomputer applications. Additionally, the language must provide those constructs of X3.23-1974 which support structured programming techniques.

Other criteria used in developing the HYPO-COBOL specifications are:

- (a) User-defined names were limited to a maximum of 12 characters.
- (b) All optional words were deleted from the language.
- (c) In the cases where the COBOL language provides both long and short forms for reserved words, the shorter form was used and the longer form deleted from the specifications.
- (d) The ellipsis was eliminated from the language formats.

The specifications contained in X3.23-1974 include language elements whose syntactic definition permit an unspecified number of repetitions of portions of the format. In most cases multiple operands are not permitted in HYPO-COBOL, and where permitted, limits on the number of repetitions were given.

(e) All functionally redundant language elements were eliminated. Only one format was included where more than one format is provided for the same function, such as the arithmetic statements. Statement phrases were deleted if they could be accomplished by other statements; for example, the GIVING phrase in arithmetic statements.

THE HYPO-COBOL LANGUAGE

HYPO-COBOL Modules

The HYPO-COBOL language consists of subsets of the Nucleus and five functional processing modules: Table Handling, Sequential I-O, Relative I-O, Debug, and Inter-Program Communication. The Relative I-O module was included since most minicomputer and time-sharing systems support some form of random access disk files. As is the case for X3.23-1974, if an implementation does not support random access devices or the necessary hardware is absent, then this module would not be required.

The subsets for the Debug and Inter-Program Communication modules

provide basic language features which can easily be implemented in a limited resources environment. The ability to CALL subprograms and the use of debug lines controlled by a compile time switch are included in the HYPO-COBOL language.

Six functional processing modules of X3.23-1974 are not included in the HYPO-COBOL language. The Indexed I-O module was deemed to be too sophisticated and demanding for a limited resources environment. The Sort-Merge module was not selected because the capability is generally available as a system utility, and the Library module functions can be handled by a text editor or similar system utility. The Segmentation module was excluded on the assumption that an overlay capability would be provided by the operating system. The Report Writer module was not considered since it requires a large amount of syntax to accomplish the function. Only limited use of the Communication module is foreseen until semantic interpretation of the module is better defined. The basic function of communicating with terminals could be handled through the ACCEPT and DISPLAY statements.

HYPO-COBOL Language Elements

Nucleus

One of the goals of the HYPO-COBOL language design was to reduce the size and complexity of COBOL compilers without greatly sacrificing the usefulness of the language. For this reason some elements from Nucleus were deleted and other elements were subset to simplify their implementation. A HYPO-COBOL compiler only has to support user-defined words of 1 to 12 characters, and the reference format rules

have been made more rigid. Division headers, section headers, paragraph headers, paragraph names and level 01 indicators must begin in the eighth character position. A nonnumeric literal may be from 1 to 50 characters in length and may not be continued across source lines.

The elements of Nucleus level 1 which are not included in the HYPO-COBOL language because of their implementation complexity are:

- ° PROGRAM COLLATING SEQUENCE clause of the OBJECT-COMPUTER paragraph;
- ° SPECIAL-NAMES paragraph;
- ° JUSTIFIED clause of a Data Description Entry;
- ° ALTER statement; and
- ° INSPECT statement.

Many redundant features contained in Nucleus level 1 of X3.23-1974 are not included in the HYPO-COBOL language. Instead of having several different formats for an arithmetic statement, there is only one format specified for each of the arithmetic statements. The GIVING phrase of an arithmetic statement has also been deleted since its functions can be accomplished by following an arithmetic statement with a MOVE statement. Other redundant features not included in HYPO-COBOL are level 77 data items which are redundant with level 01 elementary items; the BLANK WHEN ZERO clause of a Data Description Entry, which can be accomplished with zero suppression editing; and the ELSE NEXT SENTENCE phrase of the IF statement which is nothing more than documentation.

The replacement of ellipsis affected the format specifications of five Nucleus verbs. In the ADD, SUBTRACT, and DISPLAY statements the ellipsis has been replaced by a maximum of two operands. Only one receiving field is permitted in a MOVE statement and a specification limit of 20 possible procedure-names has been placed on the GO DEPENDING statement. The language specifications for these verbs are included in Figure 1.

Several Nucleus verbs have been included in the HYPO-COBOL language without any change in their format specifications. The ACCEPT, ENTER, EXIT, GO, PERFORM, PERFORM TIMES, and STOP statements were accepted without any change.

In all cases but one, the HYPO-COBOL language consists of elements from level 1 of Nucleus and the five functional processing modules subset for HYPO-COBOL. The sole exception is the UNTIL phrase of the PERFORM statement. The PERFORM UNTIL statement is included to facilitate the use of structured programming in HYPO-COBOL. The DO-WHILE and DO-UNTIL structured programming constructs require the PERFORM UNTIL statement⁶.

The I-O Modules

The HYPO-COBOL language specifications include the Sequential I-O and Relative I-O modules. These modules are proper subsets of their respective modules in X3.23-1974. Several clauses and statements were eliminated from the X3.23-1974 specifications in arriving at the HYPO-COBOL specifications.

In order to reduce the size and complexity of a HYPO-COBOL com-

piler, four major language elements were deleted. They are:

- FILE-STATUS clause of the File-Control Entry;
- RERUN clause of the I-O-CONTROL Paragraph;
- CODE-SET clause in a File Description Entry; and
- USE statement in the Procedure Division.

The choice not to select the FILE-STATUS clause and the USE statement allows compatibility with COBOL 68 as well as with COBOL 74 for the two I-O modules. In the File Description Entry, the DATA RECORD clause was deleted since it only served as documentation.

The ellipsis was replaced with a fixed maximum number of operands in the SAME clause, the OPEN statement and the CLOSE statement. The SAME clause can specify up to 5 possible file-names, and the OPEN and CLOSE statements can only contain a single file-name.

In the Procedure Division for the I-O modules, the INTO phrase was deleted from the READ statement and the FROM phrase is no longer a part of the WRITE and REWRITE statements. These phrases were omitted since their function can be accomplished by a MOVE statement following a READ statement or preceding the WRITE and REWRITE statements. Since USE procedures are not part of the HYPO-COBOL language, the END phrase is required in a sequential READ statement and the INVALID phrase must be specified in the DELETE, READ, REWRITE, and WRITE statements for files in the random access mode.

Table Handling Module

The Table Handling Module for the HYPO-COBOL language is a subset of Table Handling level 1 in X3.23-1974. HYPO-COBOL permits the

definition of one dimensional tables of like elements, and the use of a subscript to reference an individual element within a table. Indexes were eliminated from the language and therefore, the USAGE INDEX and INDEXED BY clauses are not included in the Data Description entry, and the SET statement is also not required in the Procedure Division.

Inter-Program Communication Module

The HYPO-COBOL language contains the Inter-Program Communication module level 1 from X3.23-1974. The USING phrase in the Procedure Division Header and in the CALL statement contain up to five possible operands for the passing of parameters. The data description for entries in the Linkage Section are defined according to the rules for the Working-Storage Section in the Nucleus.

Debug Module

The HYPO-COBOL language specifications include debug lines and the DEBUGGING MODE clause of the SOURCE-COMPUTER paragraph. The debug lines are source statements that are optionally compiled depending upon whether or not the DEBUGGING MODE clause is included in the SOURCE-COMPUTER paragraph when the source program is compiled. It would be unreasonable to require that debugging sections be implemented on minicomputer systems, so the USE FOR DEBUGGING statement was not selected. There is not execution time switch since debug lines are not affected by the execution time switch.

HYPO-COBOL Procedure Division Statements

In order to give the reader an idea of what the HYPO-COBOL

language is and how it differs from the specifications in American National Standard Programming Language COBOL, X3.23-1974, emphasis has been given to the elements deleted from level 1 of the modules which were subset for HYPO-COBOL. At this point some readers may feel that the HYPO-COBOL language is too restricted for use in solving application problems. The general formats for verbs given in Figure 1 should assuage that fear. There are 20 verbs in the HYPO-COBOL language, and there are more than enough features of the COBOL language included in HYPO-COBOL to make it a usable language for an applications programmer.

HYPO-COBOL COMPILER VALIDATION SYSTEM

A HYPO-COBOL Compiler Validation System (HCVS) has also been developed by the Software Development Division of the ADPE Selection Office, Department of the Navy. The HCVS permits the testing of a compiler's adherence to the HYPO-COBOL language specifications. It is similar in scope, purpose and approach to the COBOL Compiler Validation System developed by the Department of the Navy⁷.

The HYPO-COBOL Compiler Validation System consists of audit routines, their related data and an executive routine which prepares the audit routines for compilation. Each audit routine is a HYPO-COBOL program which includes specific tests of language features and the supporting procedures indicating the result of the tests. The audit routines collectively contain the features of the HYPO-COBOL language.

The executive routine is a HYPO-COBOL program which creates a

compilable program from the source program library tape. The executive routine inserts implementor names in the source code, and allows one to update an audit routine while creating the output file of the compilable source program.

The testing of a compiler supporting HYPO-COBOL in a particular hardware/operating system environment is accomplished by compiling and executing each audit routine. If the compiler rejects some language element by terminating compilation, giving fatal diagnostic messages, or terminating execution abnormally, then the statements containing the code the compiler was unable to process are deleted, and the audit routine compiled again. The output report produced by each audit routine indicates whether the compiler passed or failed the tests in the routine.

An example of a HYPO-COBOL test for the ADD statement is shown in Figure 2. If this test caused compile time problems, the test would be deleted by inserting an asterisk (*) in column 7 of the fatal test paragraph code, thereby changing the source code in the test paragraph to comment statements. The test deletion procedure paragraph ADD-DELET-36, would be executed.

The results of the compilations and the output reports for each audit routine are the raw data from which one determines whether a compiler meets the HYPO-COBOL language specifications. However, the HYPO-COBOL Compiler Validation System does not attempt to evaluate the implementation of a compiler nor its quantitative performance characteristics.

SUMMARY

The HYPO-COBOL language was prompted by the need for a COBOL subset in a small scale package that could exist in minicomputer and time-sharing environments which impose resource limitations. A COBOL subset was needed which could provide basic capabilities yet still not make extreme demands on the system resources at compile time. HYPO-COBOL is designed so as to reduce the size of the compiler and its related work areas, and to eliminate functional redundancy provided by multiple options in the language statements.

The HYPO-COBOL language is a subset of American National Standard Programming Language COBOL, X3.23-1974. Thus any program written in HYPO-COBOL conforms to the X3.23-1974 specifications and can be compiled and executed on any system which supports the appropriate levels of the X3.23-1974 Standard. HYPO-COBOL is also a subset of the Low Intermediate level of Federal Standard COBOL as defined in FIPS PUB 21-1.

The response from the COBOL community thus far has been very positive, and we are interested in receiving further comments as to the usefulness of the defined HYPO-COBOL language and suggestions for improvement in the language definitions. HYPO-COBOL gives a user in a limited resources environment a workable subset of the COBOL language for use in applications programming. Acceptance of the HYPO-COBOL language and implementation of HYPO-COBOL compilers will guarantee users portability of their COBOL programs on minicomputer systems.

Availability

The HYPO-COBOL Language Specifications are available from the National Technical Information Service; 5285 Port Royal Road; Springfield, Virginia 22161. The NITS accession number is ADA018916 and the price is \$6.25 for hardcopy and \$2.25 for microfiche. A magnetic tape containing the HYPO-COBOL Compiler Validation System will be available for distribution by NTIS in late Spring 1976.

REFERENCES

1. GML Minicomputer Review 1975, GML Corporation, Lexington, Massachusetts, 1975.
2. American National Standard Programming Language COBOL, X3.23-1974, American National Standards Institute Incorporated, New York, 1974.
3. HYPO-COBOL Language Specifications, ADPE Selection Office, Department of the Navy, Washington, D. C., 1975.
4. Federal Information Processing Standards Publication 21-1, U. S. Government Printing Office, Washington, D. C., 1975.
5. The American Heritage Dictionary of the English Language, American Heritage Publishing Company, New York, 1970.
6. Oliver, Paul, "COBOL '74 - Contributions to Structured Programming", Proc. 1975 NCC, AFIPS Press, Vol. 44, pp. 309-312.
7. Baird, G. N., "The DOD COBOL Compiler Validation System", Proc. 1972 FJCC, AFIPS Press, Volume 41, pp. 819-827.

GENERAL FORMAT FOR VERBS

ACCEPT identifier

ADD { identifier-1 } [identifier-2]
 literal-1 literal-2

TO identifier-3 [ROUNDED]

 [SIZE ERROR imperative-statement]

CALL literal [USING data-name-1 [data-name-2] ... [data-name-5]]

CLOSE file-name

DELETE file-name [INVALID imperative-statement]

DISPLAY { identifier-1 } [identifier-2]
 literal-1 literal-2

DIVIDE { identifier-1 } INTO identifier-2 [ROUNDED]
 literal

 [SIZE ERROR imperative-statement]

ENTER language-name [routine-name]

EXIT [PROGRAM]

GO procedure-name-1

GO procedure-name-1 [procedure-name-2] ... procedure-name-20

DEFENDING identifier

IF condition { imperative-statement-1
 NEXT SENTENCE } ELSE imperative-statement-2

FIGURE 1.
HYPO-COBOL Verb Formats

GENERAL FORMAT FOR VERBS

MOVE $\left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal} \end{array} \right\}$ TO identifier-2

MULTIPLY $\left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal} \end{array} \right\}$ BY identifier-2 [ROUNDED]

[SIZE ERROR imperative-statement]

OPEN $\left\{ \begin{array}{l} \text{INPUT file-name-1} \\ \text{OUTPUT file-name-2} \\ \text{I-O file-name-3} \end{array} \right\}$

PERFORM procedure-name-1 [THRU procedure-name-2]

PERFORM procedure-name-1 [THRU procedure-name-2]

$\left\{ \begin{array}{l} \text{identifier} \\ \text{integer} \end{array} \right\}$ TIMES

PERFORM procedure-name-1 [THRU procedure-name-2] UNTIL condition

READ file-name INVALID imperative-statement

READ file-name END imperative-statement

REWRITE record-name [INVALID imperative-statement]

STOP RUN
 literal

SUBTRACT $\left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \end{array} \right\}$ $\left[\begin{array}{l} \text{identifier-2} \\ \text{literal-2} \end{array} \right]$ FROM identifier-3 [ROUNDED]

[SIZE ERROR imperative-statement]

WRITE record-name $\left[\left\{ \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right\} \text{ ADVANCING } \left\{ \begin{array}{l} \text{integer} \\ \text{PAGE} \end{array} \right\} \right]$

WRITE record-name INVALID imperative-statement

FIGURE 1. (Cont.)

```
ADD-TEST-36.  
  MOVE A99-CS-02V00 TO WRK-CS-02V02.  
  ADD A99-CS-02V00 TO WRK-CS-02V02  
    SIZE ERROR PERFORM PASS  
    GO ADD-WRITE-36.  
  GO ADD-FAIL-36.  
ADD-DELET-36.  
  PERFORM DE-LETE.  
  GO ADD-WRITE-36.  
ADD-FAIL-36.  
  MOVE 'SIZE ERROR NOT EXECUTED' TO RE-MARK.  
  PERFORM FAIL.  
ADD-WRITE-36.  
  MOVE 'ADD-TEST-36' TO PAR-NAME.  
  PERFORM PRINT-DETAIL.
```

FIGURE 2.
HYPO-COBOL Test for ADD Statement

BIBLIOGRAPHIC DATA SHEET		1. Report No. FCCTS/TR-77/16	2.	3. Recipient's Accession No.
4. Title and Subtitle HYPO-COBOL -- A COBOL Subset for Minicomputers		11	5. Report Date 9 May 1977	
6.			7. Author(s) Margaret M. Cook and George N. Baird	8. Performing Organization Rept. No.
9. Performing Organization Name and Address Software Development Division ADPE Selection Office Department of the Navy Washington, D. C. 20376		12 IP.	10. Project/Task/Work Unit No.	
12. Sponsoring Organization Name and Address ADPE Selection Office Department of the Navy Washington, D. C. 20376			11. Contract/Grant No.	
13. Type of Report & Period Covered			14.	
15. Supplementary Notes				
16. Abstracts The HYPO-COBOL language is a COBOL subset designed for implementation in mini-computer and time-sharing environments which impose resource limitations. The HYPO-COBOL subset is a small scale package which could provide basic capabilities, yet still not make extreme demands on system resources during compilation. It is intended as an alternative to full COBOL for those users who do not require the complete COBOL language. The base document from which the specifications for the HYPO-COBOL language were developed was American National Standard Programming Language COBOL, X3.23-1974. The HYPO-COBOL language consists of subsets of the Nucleus and five functional processing modules; Table Handling, Sequential I-O, Relative I-O, Debug and Inter-Program Communication. This paper presents the background and reasons for the definition of the HYPO-COBOL language, and the design features and rationales of the language. A HYPO-COBOL				
17. Key Words and Document Analysis. 17a. Descriptors Compiler Validation System for determining compliance to the specifications has been developed and is also described.				
17b. Identifiers/Open-Ended Terms				
17c. COSATI Field/Group 09/02				
18. Availability Statement Release unlimited.		19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 20	
		20. Security Class (This Page) UNCLASSIFIED	22. Price	